Row-Column Beamforming with Dynamic Apodizations on a GPU

Stuart, Matthias Bo; Schou, Mikkel; Jensen, Jørgen Arendt

Published in:
Proceedings of SPIE

Link to article, DOI:
10.1117/12.2512418

Publication date:
2019

Document Version
Peer reviewed version

Row-Column Beamforming with Dynamic Apodizations on a GPU

Matthias Bo Stuart, Mikkel Schou, Jørgen Arendt Jensen
Center for Fast Ultrasound Imaging, Department of Health Technology, Technical University of Denmark

ABSTRACT

A delay-and-sum beamformer implementation for 3D imaging with row-column arrays is presented. It is written entirely in the MATLAB programming language for flexible use and fast modifications for research use, and all parts can run on either the CPU or GPU. Dynamic apodization with row-column arrays is presented and is supported in both transmit and receive. Delay calculations are simplified compared to previous beamformers, and 3D delay and apodization calculations are reduced to 2D problems for faster calculations. The performance is evaluated on an Intel Xeon E5-2630 v4 CPU with 64 GB RAM and a NVIDIA GeForce GTX 1080 Ti GPU with 11 GB RAM. A 192+192 array is simulated to image a volume of 96-by-96-by-45 wavelengths sampled at 0.3 wavelength in the axial direction and 0.5 wavelength in the lateral and elevation directions giving 5.53 million sample points. A single-element synthetic aperture sequence with 192 emissions is used. The 192 volumes are beamformed in approximately 1 hour on the CPU and 5 minutes on the GPU corresponding to a speed-up of up to 12.2 times. For a smaller beamforming problem consisting of the three center planes in the volume, a speed-up of 4.6 times is found from 109 to 24 seconds. The GPU utilization is around 5.0% of the possible floating point calculations indicating a trade-off between the easy programming approach and high performance.

1. INTRODUCTION

3D ultrasound imaging requires that the ultrasound beam can be steered in both azimuth and elevation directions. This is feasible using fully-addressed matrix arrays.[1,2] The attainable image quality in terms of resolution, contrast, and penetration depth scales with the number of elements. In 2D imaging, typically 128 to 256 elements are used in a 1D array. To attain the same image quality in 3D, thus, requires squaring the number of elements resulting in 16,384 to 65,536 individual transducer elements. Addressing these elements individually requires as many coaxial cables in the transducer cable resulting in very high system complexities and ergonomic issues. For comparison, the largest research scanners in use have only 1,024 channels,[3] while some in principle allow for even higher channel counts although no uses have been reported.[4]

An alternative is to use micro beamforming,[5] where electronics are integrated in the handle to perform part or all of the beamforming operations there. This is the main approach used in high-end commercial systems. It, however, leads to challenges with probe heating,[6] while providing lower image quality than the fully addressed array.

The number of connections can be reduced by addressing rows and columns of the matrix array rather than individual elements. This can be attained using a crossed electrode[7] structure. A modification of the delay calculation in the delay-and-sum (DAS) beamformer is necessary to account for the tall elements, and Rasmussen et al[8] presented a modified delay-and-sum beamformer for such row-column addressed (RCA) arrays. The advantage of these arrays is that the active area – translating to resolution, contrast, and penetration – scales linearly rather than quadratically with the number of connections.

This work extends the beamformer with parametric dynamic apodization in transmit and receive and presents an optimized implementation that runs on both a CPU and a GPU. The performance of the optimized implementation is reported for typical examples and the speed-up is investigated when running on the GPU compared to the CPU.

2. METHODS

This section describes the methods used. First, the principles of the MATLAB implementation are presented, then the reduction from 3D to 2D calculations is described, and finally the dynamic apodizations are described.

2.1 Code Design Principles

The beamformer is written entirely in the MATLAB programming language for high flexibility and ease of use and modification in a research setting. The GPU implementation is made by use of the `gpuArray` class in MATLAB's Parallel Computing Toolbox. This class allows variables of the integer and floating point data types to be stored in GPU memory. Most code statements including one or more `gpuArray` variables are then executed on the GPU.* No modifications to the computational part of the code is, thus, needed to choose between CPU and GPU execution.

The beamformer code is designed such that a flag called `use_gpu` is set to either true or false causing the beamformer input parameters to be allocated in either main or GPU memory. The computational part of the code is thus identical for both CPU and GPU execution. The only exception is in the interpolation, where the CPU uses the `GriddedInterpolant` class, and the GPU uses the `interp1` function. The `GriddedInterpolant` class has superior performance to `interp1` on the CPU, but is not available on the GPU.

In a system meant for clinical use, the data is most often beamformed as it is recorded, since buffering the channel data is impractical at best. This means that the beamformer setup (delay and apodization calculations or tables) change with nearly every emission. In research systems channel data is typically stored to disk for later processing. It is thus feasible to process all data acquired with the same setup in one go before moving on to the next setup. This has the advantage that the same setup is not recalculated throughout the beamforming. The beamformer is therefore written with delay and apodization calculations separated from their application to the channel data. This has the added benefit of easily extracting intermediate results and injecting e.g. experimental delay and apodization functions without needing to make changes to the base code. The trade-off is in the memory needed to store tables of delays and apodization values and the transfer of these tables from memory.

2.2 Reducing to 2D calculations

The setup calculations can be reduced to 2D even for 3D imaging. The imaging coordinate system is typically defined by the transducer array, which here consists of orthogonal line elements. By defining the coordinate system such that the x- and y-axes are parallel to the orthogonal line elements, the general 3D RCA beamformer reduces to a 2D problem.[8]

Fig. 1 illustrates a line element and an imaging point. For the line element defined by its 2 end-points

$$\mathbf{p_1} = (x_1, y_1, z_1) \text{ and } \mathbf{p_2} = (x_2, y_2, z_2) \tag{1}$$

and using the definition that this line is parallel to the y-axis (an equivalent derivation can be made for a line parallel to the x-axis), it is seen that

$$x_1 = x_2 \text{ and } z_1 = z_2. \tag{2}$$

The line segment describing the element is thus given by

$$\mathbf{l} = \mathbf{p_1} + u\mathbf{v_y} \quad , \quad u \in \begin{cases} [0; y_2 - y_1] & , \quad y_1 < y_2 \\ [y_2 - y_1; 0] & , \quad y_1 > y_2 \end{cases} , \tag{3}$$

where $\mathbf{v}_y$ is the unit vector parallel to the y-axis.

For an image point

$$\mathbf{r}_{ip} = (x_{ip}, y_{ip}, z_{ip}), \tag{4}$$

the vector

$$\mathbf{q} = \mathbf{r}_{ip} - \mathbf{p_1} \tag{5}$$

is used to find the time-of-flight T between the image point and the line element as

$$T = \frac{||\mathbf{q} \times \mathbf{v}_y||}{c}. \tag{6}$$

*A few functions do not have GPU implementations causing MATLAB to issue an error message or to copy the variables to main memory, perform the operations on the CPU, and copy the results back to GPU memory. In this work, it has been ensured that all code is executed on the GPU.

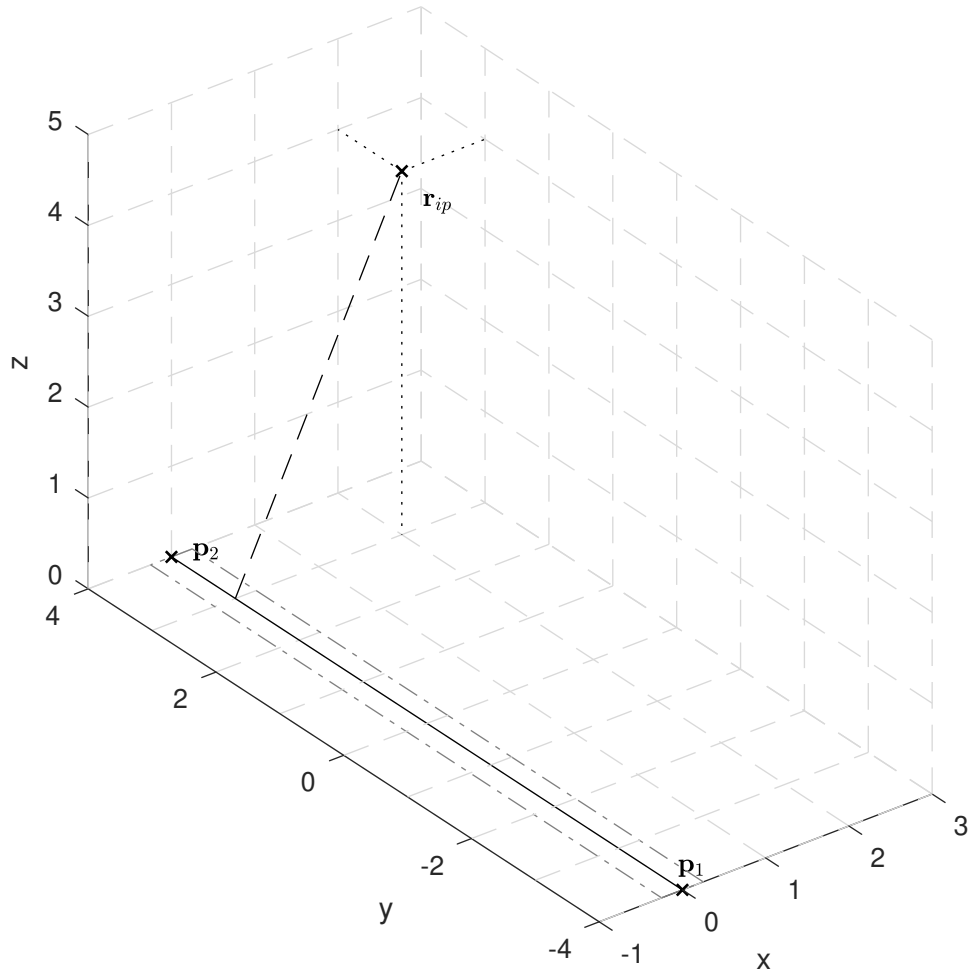

Figure 1. Illustration of the distance calculation used in the delay calculation.

Here $||\cdot||$ is the length of a vector, $\times$ is the cross-product, and c is the speed of sound. Inserting $\mathbf{v}_y$ in (6) yields

$$T = \frac{\left|\left|\mathbf{q} \times \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}\right|\right|}{c} = \frac{\left|\left|\begin{bmatrix} -q_z \\ 0 \\ q_x \end{bmatrix}\right|\right|}{c} = \frac{\left|\left|\begin{bmatrix} z_1 - z_{ip} \\ 0 \\ x_1 - x_{ip} \end{bmatrix}\right|\right|}{c} = \frac{\sqrt{(z_1 - z_{ip})^2 + (x_1 - x_{ip})^2}}{c}, \tag{7}$$

from which it is seen that the y-coordinate has been eliminated and the problem has been reduced to two dimensions.

In the row-column beamformer by Rasmussen et al,[8] different delay models are used depending on the imaging point's position relative to the line element: if the imaging point is beyond one of the line element's end-points, the distance to the end-point should be used rather than the distance to the line. The simplification above does not implement this delay model. However, the signal energy rapidly drops off when moving beyond the end-points, and no practical applications of imaging in this region have been reported. Research on imaging in this region currently focuses on applying an acoustic defocusing lens necessitating further changes to the delay calculation[9,10] and is not included in this work.

2.3 Dynamic apodization

Similarly, the dynamic apodization can be calculated in 2D. Given the same line element and imaging point as above, the relative x and z positions are found by

$$x_{rel} = x_{ip} - x_1 \quad , \quad z_{rel} = z_{ip} - z_1. \tag{8}$$

Given an F-number F, the apodization window's width, W at depth z is calculated as

$$W(z) = \frac{z}{F}, \tag{9}$$

which is used to normalize x_{rel}

$$x_{rel,norm} = \frac{x_{rel}}{W(z_{rel})}, \tag{10}$$

yielding a value that is used as input to the desired window function $w(x)$. This function accepts a normalized x-value between -0.5 and 0.5, where

$$w_{actual}(x) = \begin{cases} w(x) & \text{if } -0.5 < x < 0.5, \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

Thus, a normalized position less than -0.5 or greater than 0.5 will be outside the acceptance region defined by the F-number and the line element. For the image point $\mathbf{r}_{ip}$, the dynamic apodization value w_{dyn} of the line element containing $\mathbf{p}_1$ is thus found as

$$w_{dyn}(\mathbf{r}_{ip}, \mathbf{p}_1) = w_{actual}(x_{rel,norm}), \tag{12}$$

which is used to calculate the line element's contribution to the image point.

For an emission sequence with N emissions and M receive elements, where the nth emitting line element[†] contains the point $\mathbf{p}_{1,n}$ and the mth receive element contains the point $\mathbf{p}_{1,m}$, the value of image point $\mathbf{r}_{ip}$ is calculated as

$$v(\mathbf{r}_{ip}) = \sum_{n=1}^{N} w_{dyn,tx}(\mathbf{r}_{ip}, \mathbf{p}_{1,n}, \theta) \sum_{m=1}^{M} w_{dyn,rx}(\mathbf{r}_{ip}, \mathbf{p}_{1,m}) s_m(T), \tag{13}$$

where s_m is the signal recorded from element m, and θ is the steering angle of the transmit wave.

For the case of an emission steered at angle θ where the apodization profile should be rotated, x_{rel} and z_{rel} are rotated around the line element, such that their rotated images x'_{rel} and z'_{rel} can be substituted in the calculations above, i.e.,

$$\begin{bmatrix} x'_{rel} \\ z'_{rel} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_{rel} \\ z_{rel} \end{bmatrix}, \tag{14}$$

and

$$x'_{rel,norm} = \frac{x'_{rel}}{W(z'_{rel})}. \tag{15}$$

2.4 Evaluation setups

Point targets are simulated using Field II.[11,12] Two setups reflecting point spread function (PSF) investigations are used to benchmark the beamformer: 1) XZ, YZ, and XY planes containing the point target and 2) a volume surrounding the point target. Both setups use a 192+192 element $\lambda/2$-pitch array with 16λ roll-off apodizations.[8] Synthetic aperture imaging[13,14] is performed by emitting with a single element at a time and receiving with all elements for 192 emissions.[‡] The sampling density in the beamformer is decided by oversampling the planes in setup 1 and performing a 2D FFT. A -60 dB threshold (rel. peak) is used to choose the axial and lateral sampling densities, such that the Nyquist limit is satisfied for the highest frequency exceeding the -60 dB threshold. The

[†]This may be a virtual line element corresponding to a focused or defocused emission.

[‡]The beamformer has also been validated for synthetic aperture imaging with defocused and focused emissions.

Setup	CPU			GPU			Speed-up
1 (3 planes)	109.47	±	1.25	23.89	±	0.02	4.6
2 (volume)	3657.48	±	19.38	299.52	±	0.98	12.2
3 (5 repeats of setup 1)	521.57	±	2.57	117.51	±	0.60	4.4

Table 1. Running times in seconds and speed-up of GPU over CPU.

resulting imaging dimensions are 96λ in x- and y-directions and 45λ in the z-direction with 0.3λ sampling in the axial directions and 0.5λ in the lateral and elevation directions corresponding to $192 \times 192 \times 150 = 5.53$ million voxels. The same sampling density is used in setup 2. Dynamic apodization with a Hann window and an F-number of 1 is used in both transmit and receive in both setups. Simulation results are filtered with a matched filter including the Hilbert transform before beamforming. The beamformer, thus, processes complex data.

To evaluate the performance impact of storing delay and apodization tables for potential reuse, the first setup was rerun with the emission sequence repeated 5 times. Emissions of the same type were processed in batches, such that the number of changes to the beamformer setup were minimized.

The beamformer is evaluated on a workstation with an Intel Xeon E5-2630 v4 processor with 64 GB RAM and a NVIDIA GeForce GTX 1080 Ti graphics card with 3584 CUDA cores and 11 GB RAM. MATLAB R2017a was used on a GNU/Linux kernel 4.4.0-138 from the Ubuntu 16.04 distribution. The GPU used nVidia driver version 384.130. For the evaluation, all non-essential programs were terminated. The graphics card was also used as the display card for the system. Cubic interpolation and double precision floating point calculations are used on both CPU and GPU. The volumetric PSF requires large amounts of memory necessitating a split of the processing in two sub-volumes.

For timing measurements, the same beamforming was repeated 7 times in a loop, and all loop iterations were timed. The first 2 timing measurements were discarded to eliminate effects from e.g. GPU and cache initialization. From the remaining 5 iterations, the mean and standard deviations are reported for the different setups. The times reported are wall-clock times and include relevant changes in the beamformer setup, the interpolation, and the summing of low-resolution images. It is ensured that all GPU operations have finished before the running time is recorded.

3. RESULTS

Table 1 shows the mean and standard deviation on the running times as well as the speed-up provided by the GPU over the CPU calculated as the ratio of the mean running times.

A maximum speed-up of approximately 12 times is observed reducing the time to beamform a volume from 1 hour using the CPU to 5 minutes using the GPU.

The PSFs produced by the CPU and GPU executions are identical. A 3D PSF is shown in fig. 2, where the -6 dB, -20 dB, and -40 dB contours are shown.

4. DISCUSSION

This section presents a discussion of the results including the trade-off between the flexible interface and performance and the impact of storing intermediate results on processing time.

In setup 2, 5.53 million points are beamformed for 192 transmit events with 192 receive channels in each event. Adding up the number of operations (interpolation operation count is estimated according to the cubic spline algorithm[15]) yields 16.5 GFLOPS performance for 64-bit floating point calculations. With a 1:32 ratio of double-precision to single-precision floating point arithmetic logic units,[16] a theoretical FP64 performance of 332 GFLOPS is possible indicating approximately 5.0% utilization. This is likely a result of storing delay and apodization tables and reading these from memory making the beamformer limited by memory transfers rather than processing speed. Better performance may be attained by a parametric interface where the setup is reduced to as few parameters as possible and delay and apodization values are computed on the fly. This comes with a trade-off in reduced flexibility and has not been investigated in this work.

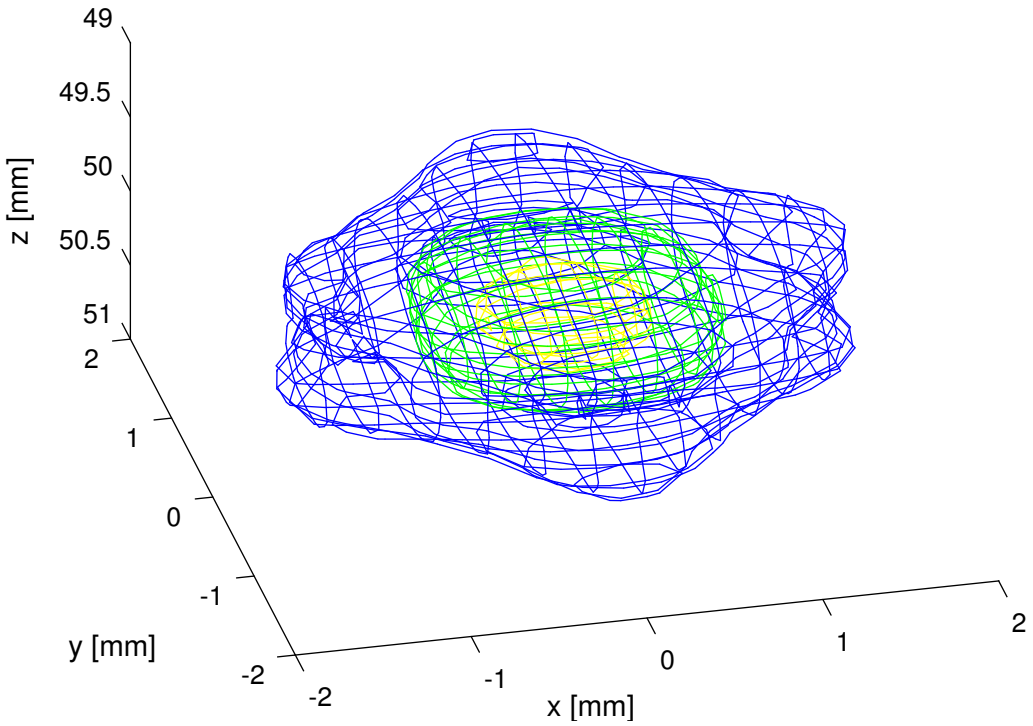

Figure 2. Contours for -6 dB (yellow), -20 dB (green), and -40 dB (blue) of the 3D point spread function from setup 2. The system's center frequency is 3.5 MHz.

Better performance may also be attained by using single precision floating point numbers instead of double precision. However, the improvement is expected to be limited to a factor 2 corresponding to the difference in the amount of data to transfer in and out of memory.

The intentions behind storing intermediate results (delays and apodizations) are 3-fold: it allows 1) easy extraction of these results for inspection and debuging of the beamformer setup, 2) easy injection of other values for researching modifications to the beamformer, and 3) faster beamforming of repeated emission setups. To investigate the performance impact of repeating the beamforming for the same setup, the beamforming and summation of low-resolution images were repeated 5 times for each emission in a variation of setup 1. As noted in Table 1, this took 117.51 seconds on average on the GPU and 521.57 seconds on the CPU. Multiplying the mean processing time of setup 1 without repetitions by 5 (the number of repetitions) yields 119.45 and 547.35 seconds respectively. These correspond to a speed-up of 1.02 and 1.05 times respectively. These are relatively small speed-ups, but often research into synthetic aperture flow imaging has processing times of weeks to months on even large high-performance computing clusters making any speed-up valuable.

5. CONCLUSIONS

A GPU implementation of the beamformer for row-column addressed arrays was presented and evaluated. The implementation includes the possibility of using focused emissions and dynamic apodization in both transmit and receive. The code was written with flexible use in research in mind and was evaluated on both CPU and GPU. A speed-up of up to 12.2 times was seen when using the GPU. However, the flexible programming approach used likely causes the code to be limited by memory throughput resulting in a low utilization of 5.0% of the processing resources.

REFERENCES

[1] S. W. Smith, H. G. Pavy, and O. T. von Ramm, "High speed ultrasound volumetric imaging system – Part I: Transducer design and beam steering," *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.* **38**, pp. 100–108, 1991.

[2] O. T. von Ramm, S. W. Smith, and H. G. Pavy, "High speed ultrasound volumetric imaging system – Part II: Parallel processing and image display," *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.* **38**(2), pp. 109–115, 1991.

[3] J. A. Jensen, H. Holten-Lund, R. T. Nilsson, M. Hansen, U. D. Larsen, R. P. Domsten, B. G. Tomov, M. B. Stuart, S. I. Nikolov, M. J. Pihl, Y. Du, J. H. Rasmussen, and M. F. Rasmussen, "SARUS: A synthetic aperture real-time ultrasound system," *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.* **60**(9), pp. 1838–1852, 2013.

[4] E. Boni, A. C. H. Yu, S. Freear, J. A. Jensen, and P. Tortoli, "Ultrasound open platforms for next-generation imaging technique development," *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.* **65**(7), pp. 1078–1092, 2018.

[5] B. J. Savord, "Beamforming method and apparatus for three-dimensional ultrasound imaging using two-dimensional transducer array." Patent US 6013032, January 2000.

[6] T. Halvorsrod, W. Luzi, and T. Lande, "A log-domain beamformer for medical ultrasound imaging systems," *IEEE Trans. Circuits Syst. I, Reg. Papers* **52**(12), pp. 2563–2575, 2005.

[7] C. E. Morton and G. R. Lockwood, "Theoretical assessment of a crossed electrode 2-D array for 3-D imaging," in *Proc. IEEE Ultrason. Symp.*, pp. 968–971, 2003.

[8] M. F. Rasmussen, T. L. Christiansen, E. V. Thomsen, and J. A. Jensen, "3-D imaging using row–column-addressed arrays with integrated apodization — Part I: Apodization design and line element beamforming," *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.* **62**(5), pp. 947–958, 2015.

[9] H. Bouzari, M. Engholm, C. Beers, M. B. Stuart, S. I. Nikolov, E. V. Thomsen, and J. A. Jensen, "Curvilinear 3-D imaging using row-column-addressed 2-D arrays with a diverging lens: Feasibility study," *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.* **64**(6), pp. 978–988, 2017.

[10] H. Bouzari, M. Engholm, C. Beers, S. I. Nikolov, M. B. Stuart, E. V. Thomsen, and J. A. Jensen, "Curvilinear 3-d imaging using row–column addressed 2-d arrays with a diverging lens: Phantom study," *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.* **65**(7), pp. 1182–1192, 2018.

[11] J. A. Jensen and N. B. Svendsen, "Calculation of pressure fields from arbitrarily shaped, apodized, and excited ultrasound transducers," *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.* **39**, pp. 262–267, 1992.

[12] J. A. Jensen, "Field: A program for simulating ultrasound systems," *Med. Biol. Eng. Comp.* **10th Nordic-Baltic Conference on Biomedical Imaging, Vol. 4, Supplement 1, Part 1**, pp. 351–353, 1996.

[13] M. Karaman, P. C. Li, and M. O'Donnell, "Synthetic aperture imaging for small scale systems," *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.* **42**, pp. 429–442, 1995.

[14] J. A. Jensen, S. Nikolov, K. L. Gammelmark, and M. H. Pedersen, "Synthetic aperture ultrasound imaging," *Ultrasonics* **44**, pp. e5–e15, 2006.

[15] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical recepies in C. The art of scientific computing*, Cambridge University Press, Cambridge, 1988.

[16] nVidia, "Pascal tuning guide." https://docs.nvidia.com/cuda/pascal-tuning-guide/index.html, 2018. v10.0.130.